CompSci 101, Fall 2015, Rubric for Exam 2

Problem 1

In general a correct answer gets full credit. Small errors are 1 point, larger 2 points.
-1 For writing a function and not assigning to the specified variable (only taken off one time for all of problem 1), or writing to some other variable name.
-0 converting int to int. string to string, list to list

Part A:

```
ecount = len([x for x in set(fruits) if x.endswith('e')])
```

Alternate Soln:

```
ecount = len([x for x in set(fruits) if x[-1] == 'e'])
```

Alternate Soln:

```
ecount = 0
for w in set(fruits):
    if w[-1] == 'e':
        ecount += 1
```

+1 use set or some equivalent to avoid dups
+1 identify last character with [-1] or endswith or .. AND COMPARES to 'e'
+1 accumulates OR uses list comprehension to collect
+1 all correct

If there's a syntactic error it's not all correct

Part B

```
bigfruit = [w for w in set(fruits) if len(w) > 7]
```

Alternate Solution:

```
bigfruit = list(set([w for w in fruits if len(w) > 7]))
```

Alternate Solution:

```
bigfruit = []
for f in set(fruits):
    if len(f) > 7:
        bigfruit.append(f)
```

Alternate Solution:

```
bigfruit = []
for f in fruits:
    if len(f) > 7:
        bigfruit += [f]
bigfruit = list(set(bigfruit))
```

+1 use list comprehension or .append to create list in context of if statement
+1 identify len element > 7 +/- 1 (so >= 7 or > 6) can get this point, > 5 cannot
+1 use of set(fruits) WHEN OTHER POINTS ABOVE ARE EARNED
+1 all correct

Part C

```
ulist = [xx for xx in fruits if 'u' in xx]
```

Alternate Solution:

```
ulist = [xx for xx in fruits if xx.find('u') >= 0]
```

Alternate Solution:

```
ulist = [xx for xx in fruits if xx.count('u') > 0]
```

Alternate Solution:

```
ulist = [xx for xx in fruits if xx.find('u') != -1 ]
```

Alternate Solution:

```
ulist = []
for f in fruits:
    if 'u' in f:
        ulist.append(f)
```

+1 identify 'u' as contained in string (get this point ***for close to correct, e.g., .find > 0***)
+1 loop over elements of f to accumulate elements in list, must have 'u' point for this
+2 all correct (see below for deductions after this earned)

-1 if use .index('u') to identify 'u' since that throws an exception if 'u' not present
-1 if use of .find('u') > 0 or other close where 'u' is argument to find

Part D

```
last = sorted(fruits)[-1]
```

Alternate Solution:

```
last = max(fruits)
```

Alternate Solution:

```
last = fruits[0]
for f in fruits:
    if f > last:
        last = f
```

+1 attempt to identify alphabetically last that's close. This could be calling sorted or using loop and comparing with > as shown
+1 consider all elements of list in process of determining last, must have previous point
+2 all correct (except for minor syntax error like sort(..) for sorted(..)

-1 for small syntax error or off-by-one, e.g., [-2] instead of [-1] to get last element.
-2 only sorted, no attempt to get last element.
-2 only got last element, no mention of sorting

Part E

```
tups = [(fruits.count(f),f) for f in set(fruits)]
st = sorted(tups)
largest = st[-1]
mostfruit = largest[1]
```

Alternative Solution, on one line:

```
mostfruit = sorted([(fruits.count(f),f) for f in set(fruits)])[-1][1]
```

Alternative Solution, using dictionary

```
d = {}
for f in fruits:
    if f not in d:
        d[f] = 1
    else:
        d[f] += 1
tups = [(value,key) for (key,value) in d.items()]
mostfruit = sorted(tups)[-1][1]
```

Alternative Solution:

```
mostfruit = ""
total = 0
for f in fruits:   # this line could also be:  for f in set(fruits):
    if fruits.count(f) > total:
        total = fruits.count(f)
        mostfruit = f
```

Scoring these solutions:

+2 create (fruit,count) tuples or sublists. These are idea points, must see attempt to count, associate count with fruit from list, and store together/link together, e.g., could even do with two parallel lists: counts and fruits with counts[i] = # fruits[i] occurrences
+2 given the previous two points, try to find largest by sorting or finding largest -- must see attempt to organize/find and access largest
+1 access largest/max with [-1] or by updating value tracked e.g., if cur > max: max = cur
+1 for trying to store string, not int, ***attempt***! but might use [0] instead of [1] above
+2 all correct (except for minor error, deduct one)
-1 small error, give the +2
-3 calculate max number, but no string

Problem 2:

Part A:

This function returns a boolean value. The expression r > g+b is boolean valued, so this returns the same thing: True when r > g+b and False otherwise.

+1 means true/false or boolean and indirectly addresses that will return same boolean value, but wording is somewhat sloppy
+2 makes clear that same value returned and that r > g+b is boolean or True/False

Part B

+1 vague reference to floating point OR int, but details aren't really clear
+2 makes clear that int values must be used for (r,g,b) pixel values and that multiplying by 1.5 creates a non-int/float that must be turned into an int

A reference to "must round" or "must truncate" should be considered an indication of an int being used

Part C

+1 references what statement does, but doesn't refer to max value of 255 for a pixel, rather says that the statement makes sure that g is less than 255, but not WHY that's important
+2 makes clear that all pixel values are maxed at 255, so that statement ensures that pixel values are kept in that range.

Part D

+1 references that isRed determines redness by whether red is greater than the sum of g and b, so clear that student has explained how isRed works
+2 makes clears that makeMoreGreen will increase the G value, and that because other values aren't changed, isRed will be true for fewer pixels. These points refer to both makeMoreGreen and it's use in conjunction with isRed

Part E

```
imred = len([x for x in img.getdata() if isRed(x)])
nimred = len([x for x in nim.getdata() if isRed(x)])
deltaRed = imred - nimred
```

alternate for one of the values, apply twice

```
imr = 0
for p in img.getdata():
    if isRed(p):
        imr += 1
```

+2 calls getdata() with loop over pixels and some check for red-ness of pixels made, but could be made incorrectly with either syntax or semantics, but clear that isRed and pixel looping are related (either for im or nim)
+2 accumulates list of red pixels with sum/len that calculates number OR increments counter or some other method for creating a variable that represents number of redpixels for one image
+1 creates two variables or expression values that each clearly represent a red pixel count, this point can only be earned if previous four points are earned
+3 all correct

Problem 3

Part A:

A dictionary of keys that are netids is created. Associated value for each key is the number of points the person with the netid as key has earned.

Purpose is to create this dictionary, same for both Solution A and B.

+1 mentions keys as netids and something about the value, but might not be clear
+1 mentions value as total of APT scores, but relationship to value not clear
+2 clear how keys and values are related

Part B:

The max value is found, then code loops over sorted keys to find the first alphabetical key that has the max value. These are lines 10 and 11 respectively

+2 mention line 10 as finding largest/max score/points earned
+2 mention that looping over sorted keys ensures that first alphabetical netid with max value is found
+2 compares this to solution A by mentioning why two if statements are used there compared to one here, this can be indirect, but must be a reference to solution A
-2 mention sorted dictionary

Part C.1:

Change line 5

```
      (net,score)=tuple(element.split(":"))
```

to

```
      (score,net) = tuple(element.split(","))
```

+1 for identifying line with either split OR tuple changed
+2 for doing whole thing correctly

Part C.2

Change lines 3-5

```
      data = item.split(",")
      net = data[1]
      score = int(data[0])
```

+1 identify the three lines and gets the split OR one of the others
+2 gets all three lines right

Part D

Change < on line 15 to >

```
      if d[akey] == mostScore and akey > mostNet:
```

+1 identify line
+3 get it right

Problem 4

Part A: Code added

```
ip = data[0]
if ip not in d:
    d[ip] = 1
else:
    d[ip] += 1
```

+1 identify ip address as key in dictionary in code created, to get this point must see taking some part of the line and use that as a key in dictionary, even if syntax all wonky
+1 key not in dictionary and key in dictionary both identified with value set and updated correctly (even if key is wrong, must see integer as value to get this point)
+2 all correct

Part B

```
def getUniqueVisits(filename):
    d = {}
    f = open(filename)
    for line in f:
        data = line.strip().split()
        ip,url = data[0], data[1]
        if url not in d:
            d[url] = [ip]
        else:
            if ip not in d[url]:
                d[url].append(ip)

    f.close()
    return d
```

Alternate Solution:
```
      def getUniqueVisits(filename):
          d = {}
          f = open(filename)
          for line in f:
              data = line.strip().split()
              ip = data[0]
              url = data[1]
              if url in d:
                  d[url].append(ip)
              else:
                  d[url] = [ip]
          f.close()
          for url in d:
              d[url] = list(set(d[url]))
          return d
```
Alternate Solution:
```
      def getUniqueVisits(filename):
          d = {}
          f = open(filename)
          for line in f:
              data = line.strip().split()
              ip = data[0]
              url = data[1]
              if url in d:
                  d[url].add(ip)
              else:
                  d[url] = set([ip])
          f.close()
          for url in d:
              d[url] = list(d[url])
          return d
```

+2 clear that in looping over lines of file, dictionary with url as key is created with some value related to the url set and updated in loop. Must see clearly the dictionary, the key, the initialize and update
+1 value is a collection of IP addresses, either a list or a set and values added to this are IP addresses
+1 Attempt to remove duplicates in values, either during creation with a list or after as shown above in creating of set and then convert to list as shown
+2 all correct (no deductions here for file issues, doesn't matter here)

-1 any file concerns, but .close() is NOT NEEDED

Part C

```
def mostDifferent(filename):
    d = {}
    f = open(filename)
    for line in f:
        data = line.strip().split()
        ip,url = data[0],data[1]
        if ip in d:
            d[ip].append(url)
        else:
            d[ip] = [url]
    values = ([(len(set(d[key])),key) for key in d])
    f.close()
    return max(values)[1]
```

Alternate Solution:

```
def mostDifferent(filename):
    d = {}
    f = open(filename)
    for line in f:
        data = line.strip().split()
        ip,url = data[0], data[1]
        if ip not in d:
            d[ip] = set()
        d[ip].add(url)
    tups = sorted([(len(val),key) for key,val in d.items()])
    f.close()
    return tups[-1][1]
```

+2 loop over lines in file, creating dictionary with ip address as key and URLs held in some collection (list or set) -- unlike previous problem, don't need to see both initialize and update, DO NEED to see loop over file, extract ip/url, and use if ip-address as key
+2 good attempt to find maximal value of tuples or related counts/ip addresses. Uniqueness isn't key here, but attempt coming during formation of dictionary, or after as shown above.
+1 sorted used correctly with tuples, e.g., count is first element of tuple, or max value of set/list tracked and updated appropriately in loop -- must see set/unique AND sort for this point
+3 all correct